Recursive Best-First AND/OR Search for Optimization in Graphical Models

Akihiro Kishimoto and Radu Marinescu
IBM Research – Ireland
{akihirok,radu.marinescu}@ie.ibm.com

Abstract

The paper presents and evaluates the power of limited memory best-first search over AND/OR spaces for optimization tasks in graphical models. We propose Recursive Best-First AND/OR Search with Overestimation (RBFAOO), a new algorithm that explores the search space in a best-first manner while operating with restricted memory. We enhance RBFAOO with a simple overestimation technique aimed at minimizing the overhead associated with re-expanding internal nodes and prove correctness and completeness of RBFAOO. Our experiments show that RBFAOO is often superior to the current state-of-the-art approaches based on AND/OR search, especially on very hard problem instances.

1 INTRODUCTION

Graphical models provide a powerful framework for reasoning with probabilistic information. These models use graphs to capture conditional independencies between variables, allowing a concise knowledge representation and efficient graph-based query processing algorithms. Combinatorial optimization tasks such as MAP or marginal MAP inference arise in many applications and are typically tackled with either search or inference algorithms (Pearl, 1988; Dechter, 2003). The most common search scheme is the depth-first branch and bound. Its use for finding exact solutions was studied and evaluated extensively in the context of AND/OR search spaces that are sensitive to the underlying problem structure (Marinescu and Dechter, 2009a,b).

Meanwhile, best-first search algorithms, despite their better time efficiency than depth-first search (Dechter and Pearl, 1985), are largely ignored in practice primarily due to their inherently enormous memory requirements (Marinescu and Dechter, 2009b). Furthermore, an important best-first search property, avoiding the exploration of unbounded paths, seems irrelevant to optimization tasks in graphical models where all solutions are at the same depth (ie, the number of variables).

We aim at inheriting the advantages of both depth-first and best-first search schemes. We introduce RBFAOO, a new algorithm that explores the context minimal AND/OR search graph associated with a graphical model in a best-first manner (even with non-monotonic heuristics) while operating within restricted memory. RBFAOO extends Recursive Best-First Search (RBFS) (Korf, 1993) to graphical models and thus uses a threshold controlling technique to drive the search in a depth-first like manner while using the available memory to cache and reuse partial search results. In addition, RBFAOO employs an overestimation method designed to further reduce the high overhead caused by re-expanding internal nodes. RBFAOO is also related to the AND/OR search algorithms based on proof/disproof numbers (Allis et al., 1994) (eg, df-pn and df-pn+ (Nagai, 2002)) which are very popular for solving two-player zero-sum games. However, since game solvers ignore the solution cost, they do not come with optimality guarantee. Moreover, while df-pn’s completeness on finding suboptimal solutions assumes that the cache table preserves all search results of nodes previously explored (Kishimoto and Müller, 2008), RBFAOO is proven to be complete with a small cache table. We evaluate empirically RBFAOO on benchmark problems used during the PASCAL2 Inference Challenge. Our results show that RBFAOO is often superior to the state-of-the-art solvers based on AND/OR search, especially on the hardest problem instances.

2 PRELIMINARIES

We consider combinatorial optimization problems defined over graphical models, including Bayesian networks and Markov random fields (Pearl, 1988; Koller and Friedman, 2009). A graphical model is a tuple $M = (X, D, F, \emptyset)$, where $X = \{X_1, \ldots, X_n\}$ is a set of variables and $D = \{D_1, \ldots, D_n\}$ is the set of their finite domains of values. $F = \{f_1, \ldots, f_r\}$ is a set of positive real-valued functions defined on subsets of variables, called scopes (ie,
∀ j f_j : Y_j → R^+ and Y_j ⊆ X). The set of function scopes implies a primal graph whose vertices are the variables and which includes an edge connecting any two variables that appear in the scope of the same function. The combination operator \( \otimes \in \{\prod, \sum\} \) defines the complete function represented by the graphical model \( M \) as \( C(X) = \otimes_{j=1}^r f_j(Y_j) \). In this paper, we focus on min-sum problems, in which we would like to compute the optimal value \( C^* \) and/or its optimizing configuration \( x^* \):

\[
C^* = C(x^*) = \min_X \sum_{j=1}^r f_j(Y_j)
\]  

(1)

Koller and Friedman (2009) convert the MAP task defined by \( \max X \prod_j f_j \) to log-space and solve it as an energy minimization (min-sum) problem to avoid numerical issues.

2.1 AND/OR SEARCH SPACES

Dechter and Mateescu (2007) introduce the concept of AND/OR search spaces for graphical models. A pseudo tree of the primal graph defines the search space and captures problem decomposition.

Definition 2.1. A pseudo tree of an undirected graph \( G = (V, E) \) is a directed rooted tree \( T = (V, E') \), such that every arc of \( G \) not included in \( E' \) is a back-arc in \( T \), namely it connects a node in \( T \) to an ancestor in \( T \). The arcs in \( E' \) may not all be included in \( E \).

A solution tree \( T \) of \( C_T \) is a subtree such that: (1) it contains the root node of \( C_T \); (2) if an internal AND node \( n \) is in \( T \) then all its children are in \( T \); (3) if an internal OR node \( n \) is in \( T \) then exactly one of its children is in \( T \); (4) every tip node in \( T \) (ie, nodes with no children) is a terminal node. The cost of a solution tree is the sum of the weights associated with its arcs.

Each node \( n \) in \( C_T \) is associated with a value \( v(n) \) capturing the optimal solution cost of the conditioned subproblem rooted at \( n \). It was shown that \( v(n) \) can be computed recursively based on the values of \( n \)'s successors: OR nodes by minimization, AND nodes by summation (see also (Dechter and Mateescu, 2007)).

Example 1. Figure 1(a) shows a simple graphical model with 5 bi-valued variables \( \{A, B, C, D, E\} \) and 3 functions \( \{f_1(ABC), f_2(ABD), f_3(BDE)\} \), respectively. Figure 2 displays the context minimal AND/OR search graph based on the pseudo tree from Figure 1(b). The contexts of the variables are shown next to the corresponding pseudo tree nodes. A solution tree corresponding to the assignment \( (A = 0, B = 1, C = 1, D = 0, E = 0) \) is highlighted.

2.2 AND/OR SEARCH ALGORITHMS

AND/OR Branch and Bound (AOBB) (Marinescu and Dechter, 2009a,b) is a state-of-the-art informed search approach for solving optimization tasks over graphical models. AOBB explores in a depth-first manner the context minimal AND/OR search graph associated with the problem and therefore takes advantage of problem decomposition. During search, AOBB keeps track of the value of the best solution found so far (an upper bound on the optimal cost) and uses this value and the heuristic function to prune away portions of the search space that are guaranteed not to contain the optimal solution in a typical branch and bound manner. Most notably, AOBB guided by a class of partitioning based heuristics won the first place in the PASCAL2 competition (Otten et al., 2012).

Best-First AND/OR Search (Marinescu and Dechter, 2009b) (AOBF) is a variant of AO* (Nilsson, 1980) applicable to graphical models that explores the graph in a best-first rather than depth-first manner. This enables AOBF to visit a significantly smaller search space than AOBB.
which sometimes translates into important time savings. Extensive empirical evaluations (Marinescu and Dechter, 2009b) showed that when given enough memory AOBF is often superior to AOBB. However, in many practical situations AOBF’s overhead of maintaining in memory the explicated portion of the search space is still prohibitively large. AOBB therefore remains the best alternative.

3 RECURSIVE BEST-FIRST AND/OR SEARCH WITH OVERESTIMATION

We introduce RBFAOO, a new algorithm that belongs to the class of recursive best-first search algorithms and employs a local threshold controlling mechanism to explore the context minimal AND/OR search graph in a depth-first like manner (Korf, 1993; Nagai, 2002). It can however use additional memory to cache and reuse partial search results to enhance performance. RBFAOO also leverages an overestimation technique to possibly find a suboptimal solution and then refine it to an optimal one. The latter plays an essential role in enhancing the performance by avoiding a high overhead of re-expanding internal nodes.

Before explaining RBFAOO in detail, we give an overview of the threshold controlling scheme that makes RBFAOO behave similarly to AO*. Assume that the weight from an OR node to an AND node is 1, the weight from an AND node to an OR node is 0, and a heuristic function \( h \) returns values as shown in Figure 3. Let \( q(n) \), called \( q \)-value, be a lower bound of the solution cost at node \( n \) and \( th(n) \) be RBFAOO’s threshold at \( n \). RBFAOO keeps examining the subtree rooted at \( n \) until either \( q(n) > th(n) \) or the subtree is solved optimally. In Figure 3(a), RBFAOO selects \( B \) to expand, because \( w(A, B) + q(B) = w(A, B) + h(B) = 3 < w(A, C) + q(C) = w(A, C) + h(C) = 5 \). It sets \( th(B) = w(A, C) + q(C) - w(A, B) = 4 \) to indicate that \( C \) becomes the best child (ie, \( w(A, B) + q(B) > w(A, C) + q(C) \) holds) if \( q(B) > th(B) \). Then, RBFAOO expands \( B \) and updates \( q(B) \) by using the q-values of \( B \)’s children (Figure 3(b)). Because \( q(B) = q(D) + q(E) = h(D) + h(E) = 3, q(B) \leq th(B) \) still holds. Hence, RBFAOO examines \( B \)’s descendants with no backtracks to \( A \). Assume that \( D \) is chosen to examine. RBFAOO sets \( th(D) = th(B) - q(E) = 2 \) to indicate that \( C \) becomes best if \( q(D) > th(D) \) holds, which is equivalent to \( q(B) > th(B) \), because \( q(B) = q(D) + q(E) \) and \( th(B) = th(D) + q(E) \). Next, RBFAOO expands \( D \) and updates \( q(D) = w(D, F) + h(F) = 4 \) (Figure 3(c)). Because \( q(D) > th(D) \), the subtree rooted at \( D \) contains no best leaf in terms of AO*’s strategy. RBFAOO backtracks to \( A \) by updating \( q(B) = q(D) + q(E) = 6 \) and examines \( C \) (Figure 3(d)) with \( th(C) = w(A, B) + q(B) - w(A, C) = 6 \) to be able to select \( B \) when \( B \) becomes best. RBFAOO gradually grows its search space by updating the q-values of internal nodes and re-expanding them. The overhead of internal node re-expansions is still high, even if RBFAOO does not always propagate back the q-values. For example, assume that an internal OR node \( n \) has two children \( c_1 \) and \( c_2 \), \( c_1 \) is selected to re-expand, and RBFAOO proves that \( c_2 \) becomes best to examine after expanding only one leaf that is \( k \)-steps away from \( c_1 \). If \( k \) is large, RBFAOO need to spend most of time in re-expanding internal nodes without exploring the new search space. The overestimation technique avoids this scenario by increasing the threshold while it verifies solution optimality.

3.1 ALGORITHM DESCRIPTION

Figure 4 shows the pseudo-code of RBFAOO. Let \( \epsilon \) be a small number and assume \( \epsilon < \infty \). In practice, a finite real number is used to represent \( \infty \). Let \( \delta \) be an empirically tuned parameter that determines the amount of overestimation. HasNoChildren checks whether a node has no children (ie, terminal leaf or dead-end) and returns a pair of the cost (ie, 0 or \( \infty \)) and a Boolean flag indicating whether \( n \) is solved or a dead-end. UnsolvedChild returns an unsolved child. SaveInCache saves in the cache table a q-value and a flag indicating whether a node is solved optimally or not. RetrieveFromCache retrieves them from the cache table. Context calculates the context of a node.

When RBFAOO starts solving a problem, the threshold of the root node is set to \( \infty - \epsilon \). If RBFAOO exceeds this threshold, the problem is proven to have no solution. Otherwise, RBFAOO returns the optimal solution cost to the problem. In addition, RBFAOO can be easily instrumented to recover the assignment corresponding to the optimal solution cost (this extension is omitted for clarity reasons).

Function \( RBFS(n) \) traverses the subtree rooted at \( n \) in a depth-first manner. It calculates either an optimal solution cost or a lower bound by using BestChild or Sum and checks if the termination condition is satisfied. If the solution optimality is guaranteed at \( n \), \( n.\text{solved} \) is set to true.

At an OR node, \( RBFS(n) \) may find a suboptimal solution. In this case, \( n.\text{solved} \) is still set to false and \( RBFS(n) \) continues examining other children until it finds an optimal solution at \( n \). Because the solution cost found so far is an upper bound of the optimal one, \( RBFS(n) \) uses that solution cost (maintained by \( ub \)) to prune away unpromising branches and to adjust the threshold.

When \( RBFS(n) \) selects \( c_{\text{best}} \), it examines \( c_{\text{best}} \) with a new threshold. At OR nodes, \( c_{\text{best}},\text{th} \) is set to subtracting the weight between \( n \) and \( c_{\text{best}} \) from the minimum of:

1. The current threshold for \( n \).
2. The second smallest lower bound \( q_2 \) to solve \( n \)’s child with considering the weight from \( n \) to that child among a list of such lower bounds of \( n \)’s children.
This indicates when the current second best child becomes the best one. Additionally, a parameter $\delta$ called overestimation rate, that allows for returning a suboptimal solution cost is added to $q_2$ to avoid an excessive number of backtracks to $n$.

3. The upper bound of the optimal solution at $n$.

At AND nodes, $c_{best}.th$ is set to the sum of $c_{best}$’s q-value and the gap between $n.th$ and the total q-value of $n$'s children. If $q(c_{best}) > c_{best}.th$, $q(n) > n.th$ also holds.

Let $N$ be the number of nodes in the search space. If the search space fits into memory, AO* expands $O(N)$ nodes in the worst case. In contrast, due to node re-expansions, RBFAOO’s worst-case scenario is $O(N^2)$. However, in practice, by introducing $\delta$, RBFAOO avoids such a high node re-expansion overhead (see Section 4).

### 3.2 IMPLEMENTATION DETAILS

The cache table is implemented as a hash table with the Zobrist function (Zobrist, 1970) using 96-bit integers. The Zobrist function computes almost uniformly distributed hash keys by XORing precomputed random integers, each of which represents the component of a context and is commonly used in game-playing programs and in planning. Each cache table entry preserves the context of a node to avoid collisions caused by an astronomically small possibility of two different nodes having the same hash key.

When RBFAOO fills up the cache table and tries to store new results there, some cached results must be replaced. We use SmallTreeGC (Nagai, 1999), a batch-based replacement that discards $R\%$ of the table entries with small subtree sizes. We set $R$ to 30.

Due to floating-point errors, RBFAOO is occasionally unable to expand a new leaf. We bypass this by using small error margins when comparing floating-point numbers.

As in full RBFS (Korf, 1993), q-values of children can be increased based on the current q-value of their parent. This technique generates more accurate heuristic information when cache table entries are replaced or non-monotonic heuristic functions are used. We implemented this technique and observed small performance improvement.

### 3.3 CORRECTNESS AND COMPLETENESS

We prove that RBFAOO is both correct and complete for solving optimization tasks defined over graphical models even with a small cache table and arbitrary cache table replacement schemes as long as certain table entries are preserved. In contrast, df-pn+ may return suboptimal solutions if the contexts of nodes are used. Additionally, although its completeness on finding either one (possibly suboptimal) or no solution is proven, df-pn+ must preserve all TT entries and the completeness with TT replacement schemes remains an open question (Kishimoto and Müller, 2008). The following theorems hold with/without enhancements described in Section 3.2.

**Theorem 3.1** (correctness). Given a graphical model $M = \langle X, D, F, \sum \rangle$, if RBFAOO solves $M$ with admissible heuristic function $h$, its solution is always optimal.

**Proof.** Let $v(n)$ be the optimal solution cost for node $n$. We first prove that for any value $q$ for node $n$ in the cache table, $q \leq v(n)$ holds. Since different nodes with the same context are proven to be equivalent in DAGs ( Marinescu and Dechter, 2009b), we denote $n$ as $\text{Context}(n)$ when a search result at node $n$ is saved in the cache table. Additionally, we assume $h(n) = h(n')$ if $\text{Context}(n) = \text{Context}(n')$.

Let $\text{Cache}_t$ be the state of the cache table immediately after the $t$-th save is performed in the cache table. Let $Q_t(n)$ be the value saved in $\text{Cache}_t$ for $n$ if that value exists in $\text{Cache}_t$ or $h(n)$ if $n$ is not preserved in $\text{Cache}_t$. By induction on $t$, we prove that all the entries in cache table contain values that do not overestimate optimal ones.

1. Because no result is stored in $\text{Cache}_0$, the above property holds for $t = 0$.

2. Assume that the above property holds for $t = k$. $Q_{k+1}(n)$, saved in $\text{Cache}_{k+1}$, is then calculated as:

   - If $n$ is a terminal leaf, Evaluate($n$) in the pseudo code always returns $v(n)$. $Q_{k+1}(n) = v(n)$ therefore holds.
\begin{itemize}
  \item If $n$ is an internal OR node, $Q_{k+1}(n) = w(n,c_{best}) + Q_k(c_{best}) = \min_i(w(n,c_i) + Q_k(c_i))$ holds where $c_i$ is $n$'s child. Additionally, because $Q_k(c_i) \leq v(c_i)$, $Q_{k+1}(n) \leq \min_i(w(n,c_i) + v(c_i)) = v(n)$ holds.
  \item If $n$ is an internal AND node, $Q_{k+1}(n) = \sum_i Q_k(c_i)$ where $c_i$ is $n$'s child. Since $Q_k(c_i) \leq v(c_i)$, $Q_{k+1}(n) \leq \sum_i v(c_i) = v(n)$ holds.
\end{itemize}

Hence, $Q_t(n) \leq v(n)$ holds in case of $t = k + 1$.

Let $Q(root)$ be a value that is about to be saved in the cache table with satisfying the termination condition of $root.solved = true$. $Q(root) \leq v(root)$ holds from the above. Additionally, because RBFAOO has traced a solution tree with the cost of $Q(root)$, $v(root) \leq Q(root)$ holds. Therefore, $Q(root) = v(root)$ holds.

Theorem 3.2 (completeness). Let $M = \langle X, D, F, \Sigma \rangle$ be a graphical model with primal graph $G$, let $T$ be a pseudo tree $G$ and let $C_T$ be the context minimal AND/OR search graph based on $T$ (also a finite DAG). Assume that RBFAOO preserves the q-values of the nodes $n_1, n_2, \cdots, n_k$ which are on the current search path and the q-values of $n_i$'s siblings. Then RBFAOO eventually returns an optimal solution or proves no solution exists.
Proof sketch. Let a marked node be a node expanded at least once by RBFAOO and an unmarked node be a node that has never been expanded. Denote \( p_1 \subset p_2 \) if the path length of \( p_2 \) is longer than that of \( p_1 \) and \( p_2 \) is identical to \( p_1 \) if \( p_2 \) is limited to the path with the length of \( p_1 \). Denote \( p_1 \not\subset p_2 \) unless it holds \( p_1 \subset p_2 \). Let \( s(n, p) \) be the sum of the edge costs from the root to \( n \) via path \( p \). Assume that \( s(n, p) < q(n) < \infty - \epsilon \) holds for any path \( p \) if \( q(n) \neq \infty \), which is reasonable in practical settings.

Let \( q_{t,p}(n) \) and \( \text{th}_{t,p}(n) \) be the q-value and threshold for \( t \)-th visit to \( n \) via path \( p \), respectively. Assume that RBFAOO expands no unmarked nodes after expanding \( k \) unmarked nodes. Then, because the search space is finite, there are two unproven marked nodes \( n \) and \( m \) examined as follows:

1. RBFAOO starts searching downward from \( n \) via path \( p_1 \) since \( q_{t_1,p_1}(n) \leq \text{th}_{t_1,p_1}(n) \) holds.
2. RBFAOO reaches \( m \) via path \( p_2 \) that satisfies \( \text{th}_{t_2,p_2}(m) < q_{t_2,p_2}(m) \) and \( p_1 \subset p_2 \).
3. RBFAOO keeps exploring the remaining search space rooted at \( n \) via \( p_1 \) (and composed of marked nodes) and backtracks to \( n \).
4. Continue steps (1)-(3).

If \( n \) is an AND node, satisfying \( \text{th}_{V'}_{p_3}(c) < q_{V'}_{p_3}(c) \) immediately leads to satisfying \( \text{th}_{t_1,p_1}(n) < q_{t_1,p_1}(n) \) where \( c \) is \( n \)'s unproven child and \( p_1 \subset p_3 \). Because backtracking to \( n \)'s parent contradicts step (1), \( n \) is an OR node. Additionally, \( \text{th}_{V'}_{p_3}(c) < \text{th}_{u,p_3}(c) \) holds for any unproven child \( c, t' < u \) and \( p_1 \subset p_3 \), because the q-values of \( n \)'s children are preserved in memory as described in the assumption of the theorem. With similar discussions, there is an infinite sequence \( u_1, u_2, \ldots, u_k \) that satisfies \( u_i < u_j \) for \( i < j \), \( \text{th}_{u_i,p_3}(m) < \text{th}_{u_j,p_2}(m) \) and \( \text{th}_{u_i,p_2}(m) < q_{u_i,p_2}(m) \). This indicates that \( m \) has at least one unproven child \( o_1 \) via path \( r_1(p_2 \not\subset r_1) \) that contributes to increasing \( q_{u_i,p_2}(m) \) and satisfying \( \text{th}_{u_i,p_2} < q_{u_i,p_2}(m) \) when \( q_{u_i,p_2}(m) \) is calculated. Because the search space is DAG, \( q(o_1) \) is never affected by \( q(m) \). With similar discussions, if no unmarked node is expanded, there is an infinite number of nodes \( o_1, o_2, \ldots, o_k, \ldots \), where \( o_{j+1} \) is a child of \( o_j \) that contributes to increasing \( q(o_j) \) and satisfying \( \text{th}(o_j) < q(o_j) \). However, this contradicts the assumption of the finite search space. Hence, by eventually examining the whole search space, RBFAOO finds an optimal solution (see Theorem 3.1) or proves no solution. \( \square \)

4 EXPERIMENTS

We empirically evaluate our proposed best-first search scheme on the MAP task in graphical models. We compare RBFAOO against the state-of-the-art depth-first and best-first AND/OR search solvers proposed recently in (Marinescu and Dechter, 2009b) and denoted by AOBF and AOBB, respectively. All competing algorithms use pre-compiled mini-bucket heuristics (Kask and Dechter, 2001; Marinescu and Dechter, 2009b) for guidance and are restricted to a static variable ordering obtained as a depth-first traversal of a minfill pseudo tree (Marinescu and Dechter, 2009a). Since AOBF cannot use an initial upper bound (obtained via local search) we also disabled its use by AOBB and RBFAOO in order to maintain a fair comparison.

Our benchmark problems\(^1\) include three sets of instances from genetic linkage analysis (Fishelson and Geiger, 2002) (denoted pedigree, grid networks and protein side-chain interaction networks (denoted pdb) (Yanover et al., 2008). In total, we evaluated 21 pedigrees, 32 grids and 240 protein networks. The algorithms were implemented in C++ (64-bit) and the experiments were run on a 2.6GHz 8-core processor with 80GB of RAM.

We report the CPU time in seconds and the number of nodes expanded for solving the problems. We also specify the problems parameters such as the number of variables \( (n) \), maximum domain size \( (k) \), the depth of the pseudo tree \( (h) \) and the induced width of the graph \( (w^*) \). The best performance points are highlighted. In each table, ‘oom’ stands for out-of-memory and ‘-’ denotes out-of-time. Note that oom for RBFAOO/AOBB indicates that the mini-bucket heuristic pre-computation procedure uses up the physical memory before search is performed.

Tables 1 and 2 show the results obtained for experiments with pedigree, grid and protein networks. For space reasons and clarity we select a representative subset from the full 293 instances. The columns are indexed by the mini-bucket \( i \)-bound which ranged between 6 and 16 for pedigrees and grids, and between 2 and 5 for proteins, respectively. All algorithms were allotted a 1 hour time limit. Algorithm AOBF(\( i \)) was allowed a maximum of 80GB of RAM while algorithm RBFAOO(\( i \)) used a 10-20GB cache table with 134,217,728 entries pre-allocated before search. The overestimation parameter \( \delta \) was set to 1.

We observe clearly that RBFAOO(\( i \)) improves considerably over its competitors, especially at relatively small \( i \)-bounds which yield relatively weak heuristics. For example, on the pedigree30 instance, RBFAOO(6) with the smallest reported \( i \)-bound \( (i = 6) \) was 4 and 41 times faster than AOBF(6) and AOBB(6), respectively. Similarly, RBFAOO(6) solves the 75-23-5 grid in about 30 minutes and expands over 300 million nodes, while both AOBB(6) and AOBF(6) run out of time and memory, respectively. As the \( i \)-bound increases and the heuristics become more accurate thus pruning the search space more effectively, the differences in running time between the algorithms de-

---

\(^1\)All instances are available at http://graphmod.ics.uci.edu.
Table 1: CPU time (seconds) and number of nodes expanded for pedigree and grid networks. Time limit 1 hour.

<table>
<thead>
<tr>
<th>instance</th>
<th>algorithm</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i (1064,42,140)</td>
<td>RBF AOO</td>
<td>2210</td>
<td>345204317</td>
<td>1368</td>
<td>216767091</td>
<td>818</td>
<td>144735023</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i (1118,7,25,123)</td>
<td>RBF AOO</td>
<td>1682</td>
<td>266815504</td>
<td>870</td>
<td>139745067</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>1084</td>
<td>159241857</td>
<td>728</td>
<td>136764248</td>
<td>532</td>
<td>97410715</td>
<td>248</td>
<td>46022921</td>
<td>241</td>
<td>44561263</td>
</tr>
<tr>
<td>i (1077,3,30,125)</td>
<td>RBF AOO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i (793,21,51)</td>
<td>RBF AOO</td>
<td>1753</td>
<td>39286572</td>
<td>834</td>
<td>166262596</td>
<td>226</td>
<td>4578797</td>
<td>378</td>
<td>69780223</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>825</td>
<td>111375519</td>
<td>1250</td>
<td>198312520</td>
<td>247</td>
<td>34132236</td>
<td>63</td>
<td>10555279</td>
<td>102</td>
<td>17743576</td>
</tr>
<tr>
<td>i (1289,5,20,105)</td>
<td>RBF AOO</td>
<td>20</td>
<td>5453997</td>
<td>19</td>
<td>5401921</td>
<td>14</td>
<td>3840692</td>
<td>5</td>
<td>1466493</td>
<td>6</td>
<td>1613986</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>307</td>
<td>9740964</td>
<td>215</td>
<td>8073776</td>
<td>53</td>
<td>2347928</td>
<td>8</td>
<td>384757</td>
<td>14</td>
<td>607660</td>
</tr>
<tr>
<td>i (1272,5,20,77)</td>
<td>RBF AOO</td>
<td>79</td>
<td>19804239</td>
<td>67</td>
<td>16206143</td>
<td>14</td>
<td>3461943</td>
<td>2</td>
<td>480866</td>
<td>4</td>
<td>666873</td>
</tr>
<tr>
<td>pedigree</td>
<td>AOBF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>i (1062,5,29,119)</td>
<td>RBF AOO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

In terms of the size of the search spaces explored, we see that AOBFi typically expands the smallest number of nodes, as expected. RBF AOOi expands more nodes than AOBFi, due to re-expansions, but in many cases it expands significantly fewer nodes than AOBFi which translates into important time savings. We also notice that RBF AOOi and AOBFi have a relatively small overhead per node expansion. On the other hand, the computational overhead of AOBFi is much larger. It is caused primarily by maintaining an extremely large search space in memory and, secondly, because the node values are typically updated all the way up to the root. Most notably, RBF AOOi was the only algorithm that could solve the most difficult instances in these benchmarks (e.g., pedigrees 7, 13, 19 and 41, as well as grid 75-26-5). This demonstrates the benefit of expanding nodes in best-first rather than depth-first manner as well as using efficiently a bounded amount of memory, thus overcoming the most critical limitation of AOBFi. Finally, the results on the protein networks show a similar pattern, namely RBF AOOi improves considerably over both AOBFi and AOBFi for relatively small i-bounds. This is important because, unlike the pedigrees and grids, these problems have very large domains (81 values) and therefore the mini-heuristic heuristics could only be compiled for small i-bounds. Figure 5 which plots the normalized total CPU time as a function of the i-bound summarizes the running time profile of the competing algorithms across the benchmarks we considered.

In Table 3 we report on five additional very difficult genetic linkage analysis networks. The mini-heuristic i-bound was set to 20 in this case. We see again that RBF AOO is the best performing algorithm closing all instances within the 100 hour time limit. In contrast, AOBFi could solve only one instance while AOBFi ran out of memory. For example, on the type4-120-17 instance, RBF AOO was nearly 3 orders of magnitude faster than AOBFi, while expanding 3 orders of magnitude fewer nodes.

We summarize next the most important additional factors that could help improve RBF AOOi’s performance.

**Impact of caching:** Table 4 shows the average performance of algorithm RBF AOOi (as CPU time in seconds, number of nodes expanded, and number of problem instances solved) as a function of available memory, across all three benchmarks. The columns are indexed by the cache table size used, namely very small (10-20MB), small...
Table 2: CPU time (seconds) and number of nodes expanded for protein networks. Time limit 1 hour. RBFAOO(i) ran with a 10-20GB cache table (134,217,728 entries) and overestimation parameter $\delta = 1$.

<table>
<thead>
<tr>
<th>instance</th>
<th>algorithm</th>
<th>$i = 2$</th>
<th>$i = 3$</th>
<th>$i = 4$</th>
<th>$i = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb1ace</td>
<td>AOBV</td>
<td>oom</td>
<td>oom</td>
<td>2218</td>
<td>6517805</td>
</tr>
<tr>
<td>(144,81,15,32)</td>
<td>RBFAOO</td>
<td>1915</td>
<td>4551</td>
<td>344</td>
<td>259261</td>
</tr>
<tr>
<td>pdb1lae</td>
<td>AOBV</td>
<td>129</td>
<td>29195</td>
<td>5</td>
<td>2694</td>
</tr>
<tr>
<td>(85,81,11,21)</td>
<td>AOBF</td>
<td>2851</td>
<td>31955</td>
<td>11</td>
<td>6264</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>81</td>
<td>1148212</td>
<td>8</td>
<td>1492</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>204</td>
<td>783</td>
</tr>
<tr>
<td>pdb1laf</td>
<td>AOBV</td>
<td>996</td>
<td>55994055</td>
<td>2672</td>
<td>16249518</td>
</tr>
<tr>
<td>(90,81,22)</td>
<td>AOBF</td>
<td></td>
<td>oom</td>
<td>16</td>
<td>1593</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>22</td>
<td>987416</td>
<td>56</td>
<td>2553896</td>
</tr>
<tr>
<td>pdb1ld2</td>
<td>AOBV</td>
<td>259</td>
<td>7770890</td>
<td>394</td>
<td>715399</td>
</tr>
<tr>
<td>(177,81,9,33)</td>
<td>AOBF</td>
<td>831</td>
<td>1250161</td>
<td>109</td>
<td>1049637</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>36</td>
<td>1227741</td>
<td>42</td>
<td>858899</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>585</td>
<td>1210780</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2543</td>
</tr>
<tr>
<td>pdb1latg</td>
<td>AOBV</td>
<td>620</td>
<td>24347033</td>
<td>4</td>
<td>71446</td>
</tr>
<tr>
<td>(175,81,12,39)</td>
<td>AOBF</td>
<td></td>
<td>38</td>
<td>119195</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>632</td>
<td>1198429</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>247</td>
<td>30872</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>256</td>
<td>11545</td>
</tr>
</tbody>
</table>

Figure 5: Normalized total CPU time as a function of the $i$-bound.

Table 3: CPU time (seconds) and node expansions for previously unsolved linkage networks. Time limit 100 hours.

<table>
<thead>
<tr>
<th>instance</th>
<th>algorithm</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>type4b-100-19 (7306,5,29,345)</td>
<td>AOBV</td>
<td>107258</td>
<td>15157422871</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>102196</td>
<td>5493591156</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>218</td>
<td>3388901</td>
</tr>
<tr>
<td>type4b-120-17 (7766,5,24,319)</td>
<td>AOBV</td>
<td>312887</td>
<td>43341893185</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>270856</td>
<td>28653407450</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>129633</td>
<td>12826083707</td>
</tr>
</tbody>
</table>

(100-200MB), medium (1-2GB) and large (10-20GB), respectively. We see that, as expected, as more memory is available, the performance improves considerably, namely more problem instances are solved while the running time and size of the search space decrease significantly. The best results were obtained with the 10-20GB cache.

Impact of overestimation: Figure 6 plots the CPU time, node re-expansion rate (as the ratio of the number of nodes re-expanded to the total number of expansions) and percentage of problem instances solved by RBFAOO(i) as a function of the overestimation rate $\delta$, across all benchmarks. We see that RBFAOO(i) without overestimation (ie, $\delta = 0$) performed rather poorly and was outperformed considerably by AOBV(i) and AOBF(i), respectively. This was due to a relatively large number of node re-expansions. On grids, for example, more than 78% of the nodes were actually re-expanded for $\delta = 0$ compared to only 11% re-expansions for $\delta = 1.2$. However, as $\delta$ increases the re-expansion rate decreases but the CPU time starts to increase due to explorations of unpromising search spaces. Therefore, we obtained the overall best performance for $\delta = 1$.

Impact of heuristics quality: Based on our empirical evaluation we noticed that RBFAOO(i) was superior to its competitors especially for relatively inaccurate heuristics (which are typically obtained for smaller $i$-bounds) and on the hardest problem instances. This is important because it is likely that for these types of problems it may only be possible to compute rather weak heuristics given limited resources (eg, type4 instances in Table 3).

5 RELATED WORK

Algorithms based on proof and disproof numbers (Allis et al., 1994) have been dominating AND/OR search techniques and successfully applied to many game domains (eg, (Nagai, 2002; Kishimoto and Müller, 2005; Schaeffer et al., 2007)). See (Kishimoto et al., 2012) for a comprehensive literature review.
Table 4: Average CPU time (seconds), number of nodes expanded and number of problem instances solved by RBFAOO(i) with different cache sizes. Time limit 1 hour. $i = 10$ for grids and pedigrees, $i = 4$ for protein networks.

<table>
<thead>
<tr>
<th>benchmark</th>
<th>10-20MB nodes solved</th>
<th>100-200MB nodes solved</th>
<th>1-2GB nodes solved</th>
<th>10-20GB nodes solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>grids</td>
<td>1928 946311520 15/32</td>
<td>1885 321943771 18/32</td>
<td>1180 223258293 14/21</td>
<td>1155 188200810 14/21</td>
</tr>
<tr>
<td>pedigree</td>
<td>1416 366052904 12/21</td>
<td>1350 27766996 13/21</td>
<td>222/240</td>
<td>1200 130087355 223/22</td>
</tr>
<tr>
<td>protein</td>
<td>574 35439167 208/240</td>
<td>509 27532421 213/240</td>
<td>443 23135969 217/240</td>
<td>396 20521511 222/240</td>
</tr>
</tbody>
</table>

A proof/disproof number estimates the difficulty of proving that the first/second player wins in a partially built tree. The proof number of node $n$ is defined as the minimum number of leaf nodes that must be expanded to prove that the first player wins at $n$. A node with a smaller proof number is assumed to be easier to prove a win for the first player. In contrast, the disproof number of $n$ is the minimum number of leaf nodes that must be expanded to prove that the second player wins at $n$. A node with a smaller disproof number is assumed to be easier to prove a win for the second player.

**Depth-First Proof-Number Search (df-pn)** (Nagai, 2002) is a depth-first formulation of Best-First Proof-Number Search (PNS) (Allis et al., 1994) enhanced with a so-called transposition table (TT), a cache table preserving the search effort for the expanded nodes. While preserving PNS’ leaf selection strategy, df-pn empirically re-expands fewer internal nodes than PNS that always restarts from the root the procedure of finding a promising leaf to expand. Besides, df-pn runs using a small amount of space limited by the TT size in practice, although whether df-pn is complete or not with a limited amount of TT still remains an open question. As in RBFS (Korf, 1993), df-pn introduces thresholds to limit the search depth of depth-first search. Df-pn updates the thresholds of a node by taking into account when the search tree rooted at that node contains none of the most promising leaf nodes chosen by PNS’ best-first strategy. The df-pn$^+$ algorithm (Nagai, 2002) generalizes df-pn by introducing evaluation functions to heuristically initialize proof and disproof numbers and a weight in each edge to decrease the overhead of node re-expansions (Kishimoto and Müller, 2005).

Other related work includes MAO* (Chakrabati et al., 1989), memory-limited AO*. Although MAO* can run under a similar memory limit to RBFAOO, it needs a specific strategy to discard examined nodes from memory. In contrast, RBFAOO can leverage arbitrary TT replacement strategies including SmallTreeGC (Nagai, 1999), which is empirically most effective in solving games. Additionally, by incorporating ideas behind RBFS and the overestimation technique, RBFAOO has much smaller overhead to update node values than MAO* and AO*.

**6 CONCLUSION**

The paper presents RBFAOO, a limited memory best-first AND/OR search algorithm for solving combinatorial optimization defined over graphical models. RBFAOO belongs to the Recursive Best-First Search family of algorithms and therefore uses a threshold controlling mechanism to guide the search in a depth-first like manner. It also employs a flexible caching scheme to reuse partial search results as well as an overestimation mechanism to further reduce the internal node re-expansions. We prove correctness and completeness of the algorithm. We evaluate RBFAOO empirically on a variety of benchmarks used during the PASCAL2 Inference Challenge. Our results show that RBFAOO is often superior to current state-of-the-art solvers based on AND/OR search, especially on the most difficult problem instances.

For future work we plan to extend RBFAOO to use dynamic variable orderings, an initial upper bound obtained via local search and soft arc-consistency based heuristics. One possibility we are currently investigating is to implement RBFAOO on top of the toulbar solver (de Givry et al., 2005). Since many interesting real-world problems are still too hard to solve exactly, we also plan to convert the algorithm into an anytime best-first search scheme.
REFERENCES


